School of Computation, Information and Technology
Technische Universität München

Lehrstuhl für
Rechnerarchitektur & Parallele Systeme
Prof. Dr. Martin Schulz
Dominic Prinz
Jakob Schäffeler

Lehrstuhl für
Design Automation
Prof. Dr.-Ing. Robert Wille
Stefan Engels

Einführung in die Rechnerarchitektur

Wintersemester 2024/2025

Übungsblatt 3: RISC-V Teil 2 – Sprünge und Pointer 04.11.2024 – 08.11.2024

1 Arrays und deren Adressierung

Felder (engl. arrays) sind ein wichtiger Bestandteil vieler Programme. Dabei werden Daten in einem linearen, zusammenhängenden Speicherbereich organisiert. Das heißt es wird ein Element nach dem anderen im Speicher abgelegt.

a) Überlegen Sie sich eine Formel mit der Sie die Adresse des n-ten Elementes in Abhängigkeit der Startadresse des Arrays für 64-Bit große Elemente ausrechnen können.

Startadresse + 8 Byte * n

b) Wie ändert sich diese Formel, wenn sich die Größe der Elemente ändert?

Startadresse + x * n

c) Im Speicher liege ein Feld von 64-Bit Zahlen. Die Startadresse des Feldes (=Pointer) stehe im Register `a0`.
Schreiben Sie ein Unterprogramm, welches das `a1`-te Element mit dem nachfolgenden tauscht. Sie dürfen annehmen, dass das Feld ausreichend lang ist. Das Startelement hat (wie in der Informatik üblich) den Index 0.

2 Zeichenketten/Strings

Auch um (nicht-unicode) Strings darzustellen, verwendet man Arrays. Um einen String abzuspeichern wird jedes Zeichen einzeln als 7-Bit Zahl kodiert und diese dann fortlaufend abgespeichert. Wir verwenden für die Kodierung einzelner Zeichen (`chars`) ASCII-Code. Das Ende eines Strings wird standardmäßig mit einem NULL-Byte gekennzeichnet (sog. C-Strings).

a) Wandeln Sie „Hallo Welt“ per Hand in ASCII um. Sie finden eine ASCII-Tabelle in Abbildung 3.

b) Ein alternativer Weg, Strings darzustellen, ist ein Längenpräfix. Diese werden auch Pascal-Strings genannt. Dabei wird die Länge des Strings explizit im ersten Byte des Strings abgespeichert.

Schreiben Sie ein Unterprogramm, welches NULL-terminierte Strings zu Pascal-Strings umwandelt. Die Startadresse des Strings werde in `a0` übergeben. Sie dürfen davon ausgehen, dass ausreichend Speicher reserviert wurde um den Pascal-String an der gleichen Speicherstelle abzuspeichern.

Bonusaufgabe: Schreiben Sie ein Unterprogramm, welches Pascal-Strings in NULL-terminierte Strings umwandelt. Die Argumente sollen gleich wie in der vorherigen Teilaufgabe übergeben werden.

c) Überlegen Sie sich mögliche Vor- und Nachteile von Pascal-Strings im Vergleich zu NULL-terminierten Strings.

Vorteile: Länge direkt ablesbar (manchmal effizienter)

Nachteile: – Länge merken
– Länge muss geupdated werden
– max Länge 255, weil 1 Byte Speicher für Länge

3 Taschenrechner-Tester (Präsenzaufgabe 01)

Bearbeitung der Präsenzaufgabe 01 (nicht bewertet) auf `https://artemis.in.tum.de/courses/401`.

4 Palindromtest (Hausaufgabe 03)

Bearbeitung und Abgabe der Hausaufgabe 03 auf `https://artemis.in.tum.de/courses/401` bis **Sonntag, den 10.11.2024, 23:59 Uhr**.

Referenzmaterial

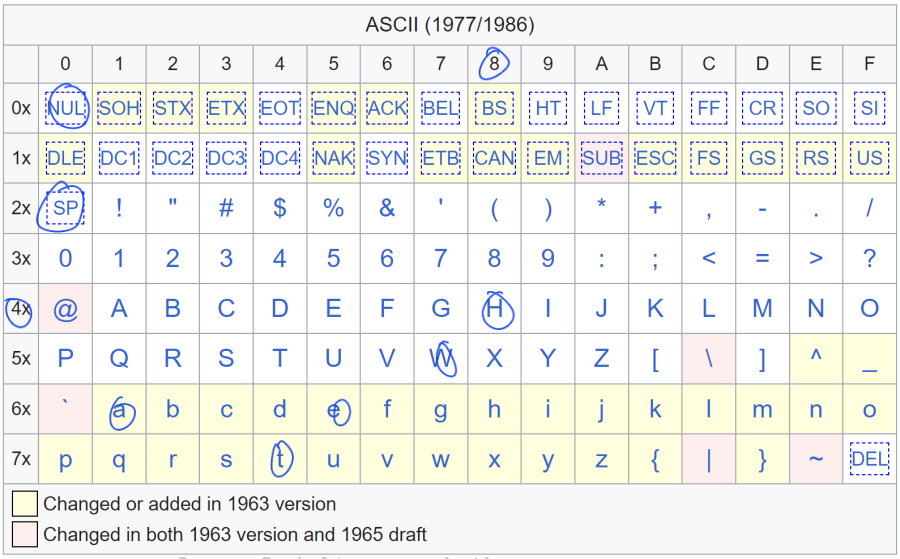
Name	Register Number	Use
zero	x0	Constant value 0
ra	x1	Return address
sp	x2	Stack pointer
gp	x3	Global pointer
tp	x4	Thread pointer
t0–2	x5–7	Temporary registers
s0/fp	x8	Saved register/Frame pointer
s1	x9	Saved register
a0–1	x10–11	Function arguments/Return values
a2–7	x12–17	Function arguments
s2–11	x18–27	Saved registers
t3–6	x28–31	Temporary registers

Abbildung 1: RISC-V 32-Bit Register

op	funct3	funct7	Type	Instruction	Description	Operation
0000011 (3)	000	–	I	lb rd, imm(rs1)	load byte	rd = SignExt([Address]$_{7:0}$)
0000011 (3)	001	–	I	lh rd, imm(rs1)	load half	rd = SignExt([Address]$_{15:0}$)
0000011 (3)	010	–	I	lw rd, imm(rs1)	load word	rd = [Address]$_{31:0}$
0000011 (3)	100	–	I	lbu rd, imm(rs1)	load byte unsigned	rd = ZeroExt([Address]$_{7:0}$)
0000011 (3)	101	–	I	lhu rd, imm(rs1)	load half unsigned	rd = ZeroExt([Address]$_{15:0}$)
0010011 (19)	000	–	I	addi rd, rs1, imm	add immediate	rd = rs1 + SignExt(imm)
0010011 (19)	001	0000000*	I	slli rd, rs1, uimm	shift left logical immediate	rd = rs1 << uimm
0010011 (19)	010	–	I	slti rd, rs1, imm	set less than immediate	rd = (rs1 < SignExt(imm))
0010011 (19)	011	–	I	sltiu rd, rs1, imm	set less than imm. unsigned	rd = (rs1 < SignExt(imm))
0010011 (19)	100	–	I	xori rd, rs1, imm	xor immediate	rd = rs1 ^ SignExt(imm)
0010011 (19)	101	0000000*	I	srli rd, rs1, uimm	shift right logical immediate	rd = rs1 >> uimm
0010011 (19)	101	0100000*	I	srai rd, rs1, uimm	shift right arithmetic imm.	rd = rs1 >>> uimm
0010011 (19)	110	–	I	ori rd, rs1, imm	or immediate	rd = rs1 | SignExt(imm)
0010011 (19)	111	–	I	andi rd, rs1, imm	and immediate	rd = rs1 & SignExt(imm)
0010111 (23)	–	–	U	auipc rd, upimm	add upper immediate to PC	rd = {upimm, 12'b0} + PC
0100011 (35)	000	–	S	sb rs2, imm(rs1)	store byte	[Address]$_{7:0}$ = rs2$_{7:0}$
0100011 (35)	001	–	S	sh rs2, imm(rs1)	store half	[Address]$_{15:0}$= rs2$_{15:0}$
0100011 (35)	010	–	S	sw rs2, imm(rs1)	store word	[Address]$_{31:0}$= rs2
0110011 (51)	000	0000000	R	add rd, rs1, rs2	add	rd = rs1 + rs2
0110011 (51)	000	0100000	R	sub rd, rs1, rs2	sub	rd = rs1 − rs2
0110011 (51)	001	0000000	R	sll rd, rs1, rs2	shift left logical	rd = rs1 << rs2$_{4:0}$
0110011 (51)	010	0000000	R	slt rd, rs1, rs2	set less than	rd = (rs1 < rs2)
0110011 (51)	011	0000000	R	sltu rd, rs1, rs2	set less than unsigned	rd = (rs1 < rs2)
0110011 (51)	100	0000000	R	xor rd, rs1, rs2	xor	rd = rs1 ^ rs2
0110011 (51)	101	0000000	R	srl rd, rs1, rs2	shift right logical	rd = rs1 >> rs2$_{4:0}$
0110011 (51)	101	0100000	R	sra rd, rs1, rs2	shift right arithmetic	rd = rs1 >>> rs2$_{4:0}$
0110011 (51)	110	0000000	R	or rd, rs1, rs2	or	rd = rs1 | rs2
0110011 (51)	111	0000000	R	and rd, rs1, rs2	and	rd = rs1 & rs2
0110111 (55)	–	–	U	lui rd, upimm	load upper immediate	rd = {upimm, 12'b0}
1100011 (99)	000	–	B	beq rs1, rs2, label	branch if =	if (rs1 == rs2) PC = BTA
1100011 (99)	001	–	B	bne rs1, rs2, label	branch if ≠	if (rs1 ≠ rs2) PC = BTA
1100011 (99)	100	–	B	blt rs1, rs2, label	branch if <	if (rs1 < rs2) PC = BTA
1100011 (99)	101	–	B	bge rs1, rs2, label	branch if ≥	if (rs1 ≥ rs2) PC = BTA
1100011 (99)	110	–	B	bltu rs1, rs2, label	branch if < unsigned	if (rs1 < rs2) PC = BTA
1100011 (99)	111	–	B	bgeu rs1, rs2, label	branch if ≥ unsigned	if (rs1 ≥ rs2) PC = BTA
1100111 (103)	000	–	I	jalr rd, rs1, imm	jump and link register	PC = rs1 + SignExt(imm), rd = PC + 4
1101111 (111)	–	–	J	jal rd, label	jump and link	PC = JTA, rd = PC + 4

*Encoded in instr$_{31:25}$, the upper seven bits of the immediate field

Abbildung 2: RISC-V 32-Bit Integerbefehle



ASCII (1977/1986)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2x	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Changed or added in 1963 version
Changed in both 1963 version and 1965 draft

"Hallo Welt\0"

48 61 6c 6c 67
20 57 65 6c 74
00

Abbildung 3: ASCII-Tabelle

